เพิ่มความสามารถให้ & ในแอปเปิ้ลซอฟท์

ชัชวาล เหลือวัฒนวรรณ

ความสามารถของภาษาเบสิกมักถูกจำกัดด้วยความเร็วในการทำงาน เพราะโปรแกรมภาษาเบสิกส่วนมากทำงานโดยใช้อินเตอร์พรีตเตอร์ แม้ว่าการใช้คอมพายเลอร์ภาษาเบสิกจะแก้ปัญหาเรื่องความเร็วได้บ้าง แต่ข้อเสียอยู่ที่การต้องหาตัวคอมพายเลอร์มาใช้และหน่วยความจำที่ต้องเสียไป เพราะโปรแกรมที่ได้รับการคอมพายล์แล้วจะมีขนาดยาวมาก เช่น โปรแกรมตัวอย่างมีบรรทัดเดียวใช้หน่วยความจำ 14 ไบท์ เมื่อทำการ

คอมพายล์แล้วใช้หน่วยความจำ 43 ไบท์ ซึ่งเป็นเฉพาะตัวโปรแกรมและยังไม่สามารถทำงานได้ ต้องมี library อีกซึ่งยาว 4013 ไบท์ รวมเป็นความยาวทั้งสิ้น 4056 ไบท์(ใช้-TASC BASIC compiler)

โปรแกรมตัวอย่าง

```
10 PRINT "HELLO"
```

อย่างไรก็ตามมีผู้คิดค้นวิธีต่าง ๆ มากมายในการเพิ่มความเร็วให้กับเบสิก วิธีหนึ่งก็คือใช้โปรแกรมย่อยต่าง ๆ ที่เป็นภาษาเครื่อง การใช้โปรแกรมย่อยเหล่านี้อาจเรียกใช้ด้วยคำสั่ง CALL หรือให้ & (ampersand) ก็ได้ แต่ข้อดีข้อเสียของทั้ง 2 วิธีก็มีต่างกันไปคือ CALL ใช้เรียกโปรแกรมย่อยใด ๆ ก็ได้ โดยเปลี่ยนตำแหน่งการ CALL ไป ส่วน & นั้นทำไม่ได้ เพราะเมื่อแอปเปิ้ลพบคำสั่ง & จะเรียกไปยังตำแหน่ง $03F5 ทันที ทำให้สามารถใช้ & ในการเรียกโปรแกรมย่อยได้เพียงตำแหน่งเดียว แต่ข้อดีของ & คือ เร็วกว่าการ CALL เพราะไม่ต้องทำการคำนวณหาตำแหน่งที่จะไปทำงาน ปัญหาคือทำอย่างไรจึงจะใช้ & ในการเรียกโปรแกรมย่อยหลาย ๆ ตำแหน่งได้ และคำตอบก็คือเขียนโปรแกรมขึ้นมาจัดการ

โปรแกรม WILDAMPER ในรูปที่ 1 เป็นตัวอย่างหนึ่งของโปรแกรมจัดการดังกล่าวที่จะทำให้สามารถใช้ & เรียกโปรแกรมย่อยได้หลาย ๆ ตำแหน่ง โดยเรียกได้มากที่สุด 256 ตำแหน่ง การเรียกโปรแกรมย่อยใช้วิธีกำหนดชื่อให้แต่ละตำแหน่ง แล้วเรียกชื่อเหล่านั้นแทนทำให้สะดวกในการเรียก ความยาวชื่อแต่ละชื่อยาวได้มากที่สุด 256 ตัวอักษร (ถ้าสามารถตั้งชื่อได้ยาวถึงขนาดนั้น) และชื่อเหล่านี้ต้องไม่เป็นคำสั่งเดิมของแอปเปิ้ล (คำสั่งเดิมเช่น LIST, READ, NEXT ฯลฯ)

วิธีการใช้โปรแกรม WILDAMPER ทำโดยการอ่านโปรแกรมเข้าสู่หน่วยความจำของแอปเปิ้ลตำแหน่งใดก็ได้ที่ว่างและมากพอ จากนั้นทำการตั้งตัวชี้ของ & ให้ชี้ไปยังจุดดังกล่าว(ตัวชี้ของ & คือ $3F5, $3F6 และ $3F7) จากนั้นก็กลับไปใช้เบสิกได้

รูปที่ 1 โปรแกรม WILDAMPER

```
                    1 ADRTBL   EQU $3A          ; จุดเริ่มต้นตารางคำสั่ง เป็นจุด
                    2 ADRTBH   EQU $3B            เก็บค่าจำนวนคำสั่ง
                    3 CMDTBL   EQU $3C          ; จุดเริ่มต้นของชื่อคำสั่ง
                    4 CMDTBH   EQU $3D            เรียงจากคำสั่งแรกไป
                    5 COUNT    EQU $40          ; เก็บค่าลำดับที่ของคำสั่งที่พบ
                    6 TEMP     EQU $41          ; หน่วยความจำสำรองใช้ในการคำนวณ
                    7 TXTPTRL  EQU $B8          ; จุดชี้ตำแหน่งของแอปเปิ้ลซอฟท์
                    8 AMPV     EQU $03F5        ; จุดเริ่มต้นของ &
                    9 ADDON    EQU $D998        ; โปรแกรมย่อยปรับค่า TXTPTR ให้ถูก
                   10 SNTX     EQU $DEC9        ; โปรแกรมย่อยพิมพ์ "SYNTAX ERRO
                   11 INCPTR   EQU $FCC2        ; โปรแกรมย่อยเพิ่มค่าใน $3C ขึ้น 1
                   12 LENGTH   EQU $0068        ; ความยาวของโปรแกรมหลัก
                   13                           ;
                   14                           ;
                   15          ORG $0300
                   16                           ;
                   17                           ;
0300: 18           18          CLC              ; คำนวณหาตำแหน่งเริ่มต้นของตารางคำ
0301: AD F6 03     19          LDA AMPV+1         แล้วนำไปเก็บไว้ใน ADRTBL และ
0304: 69 68        20          ADC #< LENGTH      ADRTBH
0306: 85 3A        21          STA ADRTBL
0308: AD F7 03     22          LDA AMPV+2
030B: 69 00        23          ADC #> LENGTH
030D: 85 3B        24          STA ADRTBH
030F: A2 00        25          LDX #$00         ; คำนวณหาจุดเริ่มต้นของตารางเก็บชื่อคำ
0311: A1 3A        26          LDA (ADRTBL,X)     แล้วนำไปเก็บไว้ใน CMDTBL และ
0313: 86 41        27          STX TEMP           CMDTBH
0315: 0A           28          ASL
0316: 26 41        29          ROL TEMP
0318: 65 3A        30          ADC ADRTBL
031A: 85 3C        31          STA CMDTBL
031C: A5 41        32          LDA TEMP
031E: 65 3B        33          ADC ADRTBH
0320: 85 3D        34          STA CMDTBH
0322: 20 C2 FC     35          JSR INCPTR
0325: 86 40        36          STX COUNT        ; เริ่มตรวจจากคำสั่งแรกในตาราง
0327: A0 00        37 LOOP1    LDY #$00         ; เริ่มการตรวจดูว่าคำสั่งที่ต้องการอยู่เป็
0329: B1 B8        38 LOOP2    LDA(TXTPTRL),Y     ลำดับที่เท่าใดในตารางคำสั่ง
032B: 41 3C        39          EOR (CMDTBL,X)
032D: 0A           40          ASL
032E: D0 26        41          BNE LOOP 3       ; ถ้าไม่ใช่คำสั่งนี้ ให้เลื่อนไป
0330: 20 C2 FC     42          JSR INCPTR       ; ถ้าพบแต่ยังไม่หมดคำให้ตรวจต่อ
0333: C8           43          INY
0334: F0 2F        44          BEQ ERR          ; ถ้าคำยาวกว่า 255 เกิด error
```

```
0336: 90 F1     45          BCC LOOP2
0338: 20 98 D9  46          JSR ADDON       ; เมื่อถึงจุดนี้แสดงว่าพบคำที่ต้องการ จากนั้น
033B: A5 40     47          LDA COUNT         คำนวณว่าหน่วยความจำที่เก็บจุดเริ่มต้น
033D: 86 41     48          STX TEMP          ของโปรแกรมย่อยดังกล่าวแล้วอ่านค่าจุด
033F: 0A        49          ASL               เริ่มต้นดังกล่าวไปเก็บไว้ในสแต็คแล้ว
0340: 26 41     50          ROL TEMP          เรียกโปรแกรมย่อยด้วย RTS
0342: 65 3A     51          ADC ADRTBL
0344: 85 3C     52          STA CMDTBL
0346: A5 41     53          LDA TEMP
0348: 65 3B     54          ADC FIDRTBH
034A: 85 3D     55          STA CMDTBH
034C: A0 02     56          LDY #$02
034E: B1 3C     57          LDA (CMDTBL),Y
0350: 48        58          PHA
0351: 88        59          DEY
0352: B1 3C     60          LDA (CMDTBL),Y
0354: 48        61          PHA
0355: 60        62          RTS
0356: A1 3C     63 LOOP 3   LDA (CMDTBL,X)  ; เลื่อน CMDTBL ให้ชี้ไปยังจุดเริ่มต้นของ
0358: 48        64          PHA               ชื่อคำสั่งถัดไป เพื่อทำการตรวจคำสั่งต่อไป
0359: 20 C2 FC  65          JSR INCPTR
035C: 68        66          PLA
035D: F0 06     67          BEQ ERR         ; ถ้าหมดชื่อคำสั่งแล้วจะเกิด error
035F: 10 F5     68          BPL LOOP3
0361: E6 40     69          INC COUNT       ; แล้วเพิ่มตัวนับเพื่อบอกว่าตรวจถึงชื่อ
0363: D0 C2     70          BNE LOOP1         คำสั่งที่เท่าไรแล้ว
0365: 4C C9 DE  71 ERR      JMP SNTX        ; พิมพ์ "SYNTAX ERROR" เมื่อเกิด error
0368: 03        72 CMDNUM   HEX 03          ; มี 3 คำสั่งในตารางคำสั่ง
0369: CF F8     73          HEX CFF8        ; จุดเริ่มต้น -1 ของโปรแกรมย่อยที่ 1
036B: D9 FD     74          HEX D9FD        ; จุดเริ่มต้น -1 ของโปรแกรมย่อยที่ 2
036C: 5F FB     75          HEX 5FFB        ; จุดเริ่มต้น -1 ของโปรแกรมย่อยที่ 3
036F: 49 4E 53  76          HEX 494E53      ; ชื่อคำสั่งแรก "INSTDSP"
0372: 54 44 53  77          HEX 544453
0375: D0        78          HEX D0
0376: 50 52 42  79          HEX 505242      ; ชื่อคำสั่งที่ 2 "PRBYTE"
0379: 59 54 C5  80          HEX 5954C5
037C: 41 50 50  81          HEX 415050      ; ชื่อคำสั่งที่ 3 "APPLE"
037F: 4C C5     82          HEX 4CC5
0381: 00        83          HEX 00          ; บอกจุดสิ้นสุดตารางชื่อคำสั่ง
                84          END
END ASSEMBLY--
130 BYTES
```

ตามปกติ เช่น ทำการอ่านโปรแกรมเข้าไว้ที่ $300 และตั้งตัวชี้ดังนี้

$3F5 ใส่ค่า $4C คำสั่งให้กระโดดการทำงาน

$3F6 ใส่ค่า $00 low byte ของตำแหน่งที่จะกระโดดไป

$3F7 ใส่ค่า $03 high byte ของตำแหน่งที่จะกระโดดไป

ในโปรแกรม WILDAMPER ที่ให้มานี้มีคำสั่ง 3 คำสั่งคือ INSTDSP, PRBYTE และ APPLE ซึ่งเป็นชื่อของโปรแกรมย่อยใน ROM ของแอปเปิ้ล

การเรียกโปรแกรมย่อยเหล่านั้นทำได้โดยใช้ & แล้วตามด้วยชื่อของคำสั่ง เช่น ต้องการเรียกโปรแกรมย่อยชื่อ APPLE ก็ใช้คำสั่ง & APPLE ดังโปรแกรม DEMO

โปรแกรม DEMO

```
10 & APPLE :LIST :PRINT :PRINT :& INSTDSP
20 PRINT :PRINT :& PRBYTE :PRINT :PRINT
   :LIST
```

นอกจากนี้ยังสามารถเปลี่ยนแปลงจำนวนคำสั่ง และเปลี่ยนชื่อแต่ละคำสั่งได้โดยไม่ยุ่งยากอีกด้วย ซึ่งรายละเอียดจะกล่าวถึงต่อไป

ส่วนสำคัญของโปรแกรมแบ่งได้เป็น 3 ส่วนคือ

1. โปรแกรม เป็นส่วนที่จะทำการตรวจคำสั่งและเรียกโปรแกรมย่อย มีความยาว 104 หรือ $68 ไบท์

2. ตารางแอดเดรส อยู่ต่อจากส่วนที่ 1 เป็นตารางเก็บตำแหน่งเริ่มต้นของโปรแกรมย่อย ลบด้วย 1 ของแต่ละโปรแกรมย่อยเรียงจากของโปรแกรมย่อยที่ 1 ถึงโปรแกรมย่อยสุดท้าย เช่น โปรแกรมย่อยเริ่มต้นที่ $6000 จะต้องเก็บค่า $6000-1 คือ $5FFF การเก็บค่าดังกล่าวจะเก็บ low byte ก่อนแล้วตามด้วย high byte ความยาวของส่วนนี้เท่ากับ 2N+1 ไบท์ เมื่อ N คือจำนวนโปรแกรมย่อยที่เรียกได้จากโปรแกรมนี้ รูปที่ 2 เป็นตารางอธิบายถึงความหมายของตำแหน่งต่าง ๆ

ไบท์	คำอธิบาย
00	จำนวนโปรแกรมย่อย หรือจำนวนคำสั่งที่มีในโปรแกรม
01-20	ตำแหน่งเริ่มต้นของโปรแกรมย่อย ลบด้วย 1 ของโปรแกรมย่อยที่ 1
03-04	ตำแหน่งเริ่มต้นของโปรแกรมย่อย ลบด้วย 1 ของโปรแกรมย่อยที่ 2
.....	.....
(2N-1)-2N	ตำแหน่งเริ่มต้นของโปรแกรมย่อย ลบด้วย 1 ของโปรแกรมย่อยที่ N

รูปที่ 2 แสดงรูปแบบของตารางแอดเดรส

3. ตารางคำสั่ง อยู่ต่อจากส่วนที่ 2 เป็นตารางเก็บชื่อของโปรแกรมย่อยแต่ละตำแหน่งเรียงตามลำดับตั้งแต่ชื่อของโปรแกรมย่อยที่ 1 ถึงโปรแกรมย่อยสุดท้าย

การเก็บชื่อแต่ละโปรแกรมย่อยจะเก็บในรูป ASCII ของแต่ละตัวอักษรของชื่อคำสั่ง โดยอักษรตัวสุดท้ายของคำสั่งจะมีบิทที่ 7 ON หรือมีค่าเป็น 1 ส่วนอักษรตัวอื่น ๆ มีบิทที่ 7 OFF หรือมีค่าเป็น 0 เช่น APPLE จะมีรหัสคือ H41 50 50 4C C5

เมื่อจบชื่อของคำสั่งหนึ่งแล้วก็ต่อด้วยอักษรแรกของคำสั่งถัดไป และเมื่อหมดอักษรสุดท้ายของคำสั่งสุดท้ายแล้วต้องใส่ค่า 00 เพื่อเป็นตัวบอกว่าจบตารางคำสั่งเพียงเท่านี้

ลำดับที่ของชื่อคำสั่งในส่วนนี้ต้องสัมพันธ์กับลำดับที่ของตำแหน่งเริ่มต้นลบด้วย 1 ในส่วนที่ 2 ด้วย คือถ้าชื่อคำสั่งอยู่เป็นลำดับที่ 3 ในตารางคำสั่ง ค่าตำแหน่งเริ่มต้นลบด้วย 1 ของโปรแกรมย่อยนั้น ก็ต้องอยู่เป็นลำดับที่ 3 ในตารางแอดเดรสด้วย

เมื่ออ่านมาถึงจุดนี้แล้วผู้อ่านคงสามารถเพิ่ม ลด หรือเปลี่ยนแปลงชื่อคำสั่งได้แล้ว เช่น ถ้าต้องการเพิ่มคำสั่งเข้าไปในตารางอีก 1 คำสั่ง คือ เพิ่ม COUT1 ซึ่งมีจุดเริ่มต้นอยู่ที่ $FDF0 จะต้องทำดังนี้

1. เปลี่ยนค่าไบท์ 00 ของตารางแอดเดรสจาก $03 เป็น $04

2. คำนวณค่าตำแหน่งเริ่มต้นลบด้วย 1 ซึ่งจะนำไปเก็บในตารางแอดเดรส จะได้ $FDEF

3. แปลชื่อ COUT1 ให้อยู่ในรูปรหัส ASCII ได้ดังนี้ 43 4F 55 54 B1 (HEX)

4. นำรหัส ASCII ที่ได้ไปต่อท้ายอักษรตัวสุดท้ายในตารางคำสั่งในตัวอย่างนี้คือนำไปใส่ไว้ที่ตำแหน่ง $0381 จากนั้นนำค่า ØØ ไปต่อท้ายอักษรตัวสุดท้ายของ COUT1 คือต่อท้าย $B1

5. เลื่อนส่วนของตารางคำสั่งทั้งหมดถอยไป 2 ตำแหน่งคือ เดิมในตัวอย่างเริ่มที่ $036F ก็เปลี่ยนไปเริ่มต้นที่ $0371

6. นำค่าตำแหน่งเริ่มต้นลบ 1 ที่คำนวณได้ไปเก็บไว้ในช่องว่างที่เกิดขึ้นโดยเก็บ low byte ก่อน แล้วตามด้วย high byte คือเก็บที่ $036F และ $0370

เมื่อทำเสร็จแล้ว จะมีการเปลี่ยนแปลงตั้งแต่ตำแหน่ง $036F เป็นต้นไปดังนี้

```
$036F: EF FD49 4E 53 54 44
$0376: 53 D0 50 52 42 59 54
$037C: C5 41 50 50 4C C5 43
$0383: 4F 55 54 B1 00
```

และตำแหน่ง $0368 เปลี่ยนจาก $03 เป็น $04

สำหรับการลดจำนวนคำสั่ง หรือการเปลี่ยนชื่อคำสั่งผู้อ่านคงทำได้ด้วยตนเอง

หลักการทำงานของส่วนโปรแกรมแบ่งได้เป็น 3 ส่วนคือ

1. ส่วนคำนวณตำแหน่งเริ่มต้น โปรแกรม WILDAMPER นี้เป็น RELOCATABLE เพื่อให้สะดวกในการใช้คือ สามารถนำไปไว้ที่ตำแหน่งใดก็ได้ จึงต้องมีการคำนวณว่าจุดเริ่มของตารางแอดเดรส และตารางคำสั่งอยู่ที่ไหน

การคำนวณจะเริ่มจากการอ่านค่าตำแหน่งเริ่มต้นของโปรแกรมจากตัวชี้ของ & จากตำแหน่ง $3F6 และ 3F7 จากนั้นนำมาบวกด้วยความยาวของส่วนโปรแกร
จะได้จุดเริ่มต้นของตารางแอดเดรส จาก
นำค่าที่ได้มาบวกด้วยความยาวของส่
ตารางแอดเดรสซึ่งมีค่า 2N+1 เมื่อ N เ
จำนวนโปรแกรมย่อย ผลที่ได้เป็นจุดเริ่ม
ของตารางคำสั่ง

2. ส่วนตรวจคำสั่ง ทำการตรวจ
คำสั่งที่เรียกมีอยู่ในตารางคำสั่งหรือไม่ แ
ถ้ามีจัดเป็นลำดับที่เท่าไร โดยเริ่มตรวจ
ลำดับแรกถึงลำดับสุดท้าย ถ้าไม่พบแสด
เกิด error

หลักการตั้งชื่อคำสั่งง่าย ๆ คือ
ใช้เป็นตัวแปรในแอปเปิ้ลซอฟท์เบสิกได้
สามารถใช้เป็นชื่อคำสั่งได้ และคำที่ใช้
ชื่อคำสั่งแล้วยังใช้เป็นชื่อตัวแปรได้อีก เพ
ชื่อคำสั่งจะมีผลเฉพาะเมื่ออยู่หลัง & เท่า

3. ส่วนเรียกโปรแกรมย่อย เมื่อท
ว่าคำสั่งที่เรียกอยู่ลำดับที่เท่าไรในตา
คำสั่งแล้ว ก็ทำการอ่านค่าตำแหน่งเริ่ม
ลบด้วย 1 ของโปรแกรมย่อยดังกล่าว
เก็บไว้ในสแต็ก โดยเก็บ high byte ก่อน
ตามด้วย low byte แล้วใช้ RTS เป็น
เรียกโปรแกรมย่อย

เหตุที่สามารถใช้ RTS ในการเ
โปรแกรมย่อยได้ เพราะเมื่อแอปเปิ้ลพบค
RTS ซึ่งเป็นคำสั่ง "กลับจากโปรแกรมย่อ
เครื่องจะทำการอ่านค่าจากสแต็กมา 2
ค่าแรกนำมาเป็น low byte ค่าต่อมานำ
ทำเป็น high byte แล้วเพิ่มค่าดังกล่าว
1 ได้เป็นแอดเดรสที่จะไปทำงานต่อ
ทำเช่นนี้เปรียบเหมือนเราหลอกเครื่อ
ได้มีการเรียกโปรแกรมย่อย จากจุดที่
หลอกไว้ (ทั้งที่ไม่ได้มีการเรียกโปรแก
ย่อยใด ๆ เลย) ดังนั้นเมื่อเครื่องพบกับค
RTS จึงทำการ "กลับจากโปรแกรมย่อ
แต่ไม่ได้กลับไปยังตำแหน่งที่ถูกต้อง
กลับไปยังจุดที่เราหลอกไว้ ซึ่งคือโปรแก
ย่อยที่เราต้องการนั่นเอง และคำสั่ง R
ของโปรแกรมย่อยที่เราเรียก จะทำการ "
จากโปรแกรมย่อย" อีกครั้งหนึ่ง คร
เป็นการกลับจากการใช้ & ไปเข้าเบสิกตาม